Does Continual Learning Equally Forget All Parameters?

Haiyan Zhao [1] **Tianyi Zhou** [2] **Guodong Long** [1] **Jing Jiang** [1] **Chengqi Zhang** [1]

Abstract

Distribution shift (e.g., task or domain shift) in continual learning (CL) usually results in catastrophic forgetting of previously learned knowledge. Although it can be alleviated by repeatedly replaying buffered data, the every-step replay is time-consuming. In this paper, we study which modules in neural networks are more prone to forgetting by investigating their training dynamics during CL. Our proposed metrics show that only a few modules are more task-specific and sensitive to task change, while others can be shared across tasks as common knowledge. Hence, we attribute forgetting mainly to the former and find that finetuning them only on a small buffer at the end of any CL method can bring non-trivial improvement. Due to the small number of finetuned parameters, such "Forgetting Prioritized Finetuning (FPF)" is efficient in computation. We further propose a more efficient and simpler method that entirely removes the every-step replay and replaces them by only k-times of FPF periodically triggered during CL. Surprisingly, this "k-FPF" performs comparably to FPF and outperforms the SOTA CL methods but significantly reduces their computational overhead and cost. In experiments on several benchmarks of class- and domain-incremental CL, FPF consistently improves existing CL methods by a large margin, and k-FPF further excels in efficiency without degrading the accuracy. We also empirically studied the impact of buffer size, epochs per task, and finetuning modules on the cost and accuracy of our methods.

[1]University of Technology Sydney [2]University of Maryland. Correspondence to: Haiyan Zhao <Haiyan.Zhao-2@student.uts.edu.au>, Tianyi Zhou <tianyi@umd.edu>, Guodong Long, Jing Jiang, Chengqi Zhang <{guodong.long, jing.jiang, chengqi.zhang}@uts.edu.au>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

1. Introduction

Empowered by advancing deep learning techniques and neural networks, machine learning has achieved unprecedented promising performance on challenging tasks in different fields, mostly under the i.i.d. offline setting. However, its reliability and performance degenerate drastically in continual learning (CL) where the data distribution or task in training changes over time, as the model quickly adapts to a new task and overwrites the previously learned weights. This leads to a severe bias toward more recent tasks and "catastrophic forgetting" of previously learned knowledge, which is detrimental to a variety of practical applications.

A widely studied strategy to mitigate forgetting is experience replay (ER) (Ratcliff, 1990; Robins, 1995) and its variants (Riemer et al., 2018; Buzzega et al., 2020; Boschini et al., 2022), which store a few data from previous tasks in the limited memory and train the model using both the current and buffered data. However, they only bring marginal improvements when the memory is too small to store sufficient data for recovering previously learned knowledge, which is common due to the complicated distributions of previous tasks. In contrast, multi-task learning (Caruana, 1997) usually adopts a model architecture composed of a task-agnostic backbone network and multiple task-specific adapters on top of it. While the backbone needs to be pre-trained on large-scale data, the adapters are usually lightweight and can be achieved using a few data. In CL, however, we cannot explicitly pre-define and separate the task-agnostic parts and task-specific parts. Although previous methods (Schwarz et al., 2018; Zenke et al., 2017) have studied to restrict the change of parameters critical to previous tasks, such an extra constraint might degrade the training performance and discourage task-agnostic modules from capturing shared knowledge.

In this paper, we study a fundamental but open problem in CL, i.e., are most parameters task-specific and sensitively changing with the distribution shift? Or is the catastrophic forgetting mainly caused by the change in a few task-specific parameters? It naturally relates to the plasticity-stability trade-off in biological neural systems (Mermillod et al., 2013): more task-specific parameters improve the plasticity but may cause severe forgetting, while the stability can be improved by increasing parameters shared across tasks.

In addition, how many task-specific parameters suffice to achieve promising performance on new task(s)? Is every-step replay necessary?

To answer these questions, we investigate the training dynamics of model parameters during the course of CL by measuring their changes over time. For different CL methods training on different neural networks, we consistently observe that **only a few parameters change more drastically than others between tasks**. The results indicate that most parameters can be shared across tasks, and **we only need to finetune a few task-specific parameters to retain the previous tasks' performance**. Since these parameters only cover a few layers of neural networks, they can be efficiently and accurately finetuned using a small buffer.

Our empirical studies immediately motivate a simple yet effective method, "*forgetting prioritized finetuning* (FPF)" which finetunes the task-specific parameters using buffered data at the end of CL methods. Surprisingly, on multiple datasets, FPF consistently improves several widely-studied CL methods and substantially outperforms a variety of baselines. Moreover, we extend FPF to a **more efficient lazy replay method** "k-FPF" that eliminates the cost of every-step replay by replacing such frequent replay with occasional FPF. k-FPF applies FPF only k times during CL. We show that a very small k suffices to enable k-FPF to achieve comparable performance with that of FPF+state-of-the-art (SOTA) CL methods and meanwhile significantly reduces the computational cost. In addition, we explore different groups of parameters to finetune in FPF and k-FPF by ranking their sensitivity to task shift evaluated in the empirical studies. For FPF, we compare them under different choices for the buffer size, the number of epochs per task, the CL method, and the network architecture. FPF can significantly improve existing CL methods by only finetuning $\leq$ **1.13%** parameters. For k-FPF, we explore different groups of parameters, k, and the finetuning steps per FPF. k-FPF can achieve a promising trade-off between efficiency and performance. Our experiments are conducted on a broad range of benchmarks for class- and domain-incremental CL in practice, e.g., medical image classification and realistic domain shift between image styles.

2. Related Work

Continual Learning and Catastrophic Forgetting A line of methods stores samples of past tasks to combat the forgetting of previous knowledge. ER (Riemer et al., 2018) applies reservoir sampling (Vitter, 1985) to maintain a memory buffer of uniform samples over all previous tasks. MIR (Aljundi et al., 2019) proposes a new strategy to select memory samples suffering the largest loss increase induced by the incoming mini-batch, so those at the forgetting boundary are selected. RAR (Kumari et al., 2022) synthesizes adversarial samples to refine the boundary between tasks and reduces bias towards the current task. DER and DER++ (Buzzega et al., 2020) apply knowledge distillation to mitigate forgetting by storing the output logits for buffered data during CL. iCaRL (Rebuffi et al., 2017) selects samples closest to the representation mean of each class and trains a nearest-mean-of-exemplars classifier to preserve the class information of samples. Our methods are complementary techniques to these memory-based methods. It can further improve their performance by finetuning a small portion of task-specific parameters on buffered data once (FPF) or occasionally (k-FPF).

Another line of work imposes a regularization on model parameters or isolates task-specific parameters to retain previous knowledge. oEWC (Schwarz et al., 2018) constrains the update of model parameters important to past tasks by a quadratic penalty. To select task-specific parameters, SI (Zenke et al., 2017) calculates the effect of the parameter change on the loss while MAS (Aljundi et al., 2018) calculates the effect of the parameter change on the model outputs when each new task comes. PackNet (Mallya & Lazebnik, 2018) and HAT (Serra et al., 2018) iteratively assign a subset of parameters to consecutive tasks via binary masks. All these works try to identify critical parameters for different tasks during CL and restrict the update of these parameters. But they can also prevent task-agnostic parameters from learning shared knowledge across tasks. From the training dynamics of CL, we identify the parameters sensitive to distribution shift. FPF and k-FPF finetune these parameters to mitigate bias without restricting the update of task-agnostic parameters.

Different modules in neural networks Pham et al. (2022) and Lesort et al. (2021) only study the effect of different normalization layers and classifiers on CL in a given setting, while our method investigates the sensitivity of all parameters in different network architectures and scenarios. Wu et al. (2022) study the forgetting of different blocks in the pre-trained language models by investigating their representation ability. We provide a more fine-grained analysis of the forgetting of each module by their training dynamics. And we find that parameters of different kinds of modules have different sensitivities to forgetting. Ramasesh et al. (2020) show that freezing earlier layers after training the first task has little impact on the performance of the second task. This is because their unfrozen part covers the last FC layer and many BN parameters, which are the most sensitive/critical according to our empirical study. Zhang et al. (2019) find that in different architectures, the parameters in the top layers (close to input) are more critical, and perturbing them leads to poor performance. Our empirical study is consistent with their findings in that the earlier convolutional layer is sensitive to task drift and the induced biases on them lead to catastrophic forgetting.

3. Problem Setup

Notations We consider the CL setting, where the model is trained on a sequence of tasks indexed by $t \in \{1, 2, \ldots, T\}$. During each task t, the training samples (x, y) (with label y) are drawn from an i.i.d. distribution D_t. Given a neural network $f_\Theta(\cdot)$ of L layers with parameter $\Theta = \{\theta_\ell\}_{\ell=1:L}$, $\theta_\ell = \{\theta_{\ell,i}\}_{i=1:n_\ell}$ denote all parameters in layer-ℓ where $\theta_{\ell,i}$ denotes parameter-i. On each task, $f_\Theta(\cdot)$ is trained for N epochs. We denote all parameters and the layer-ℓ's parameters at the end of the n-th epoch of task t by Θ_n^t and $\theta_{\ell,n}^t$, $n \in \{1, \ldots, N\}$, respectively.

Settings In this paper, we mainly focus on class-incremental learning (class-IL) and domain-incremental learning (domain-IL). In class-IL, D_t are drawn from a subset of classes C_t, and $\{C_t\}_{t=1}^T$ for different tasks are assumed to be disjoint. Class-IL is a more challenging setting in CL (Van de Ven & Tolias, 2019) than task-incremental learning (task-IL) (Lopez-Paz & Ranzato, 2017). Unlike task-IL, class-IL cannot access the task label during inference and has to distinguish among all classes from all tasks. In domain-IL, tasks to be learned remain the same, but the domain varies, i.e. the input data distribution D_t changes. The model is expected to adapt to the new domain without forgetting the old ones. The goal of the class-IL and domain-IL is: $\min_\Theta L(\Theta) \triangleq \sum_{t=1}^T \mathbb{E}_{(x,y)\sim D_t}[l(y, f_\Theta(x))]$, where l is the objective function.

Datasets We conduct class-IL experiments on Seq-MNIST, Seq-OrganAMNIST, Seq-PathMNIST, Seq-CIFAR-10, and Seq-TinyImageNet. Seq-OrganAMNIST and Seq-PathMnist are generated by splitting OrganAMNIST or PathMNIST from MedMNIST (Yang et al., 2021), a medical image classification benchmark. CL on medical images is essential in practice but also challenging since medical images always come as a stream with new patients and new diseases. Moreover, medical images of different classes might only have subtle differences that are hard to distinguish. Both Seq-OrganAMNIST and Seq-PathMnist consist of 4 disjoint classification tasks. The number of classes per task in Seq-OrganAMNIST and Seq-PathMnist are [3, 3, 3, 2] and [2, 2, 2, 2], respectively. Seq-MNIST (Seq-CIFAR-10) are generated by splitting the 10 classes in MNIST (LeCun et al., 1998) (CIFAR-10 (Krizhevsky et al., 2009)) into five binary classification tasks. Seq-TinyImageNet partitions the 200 classes of TinyImageNet (Le & Yang, 2015) into 10 disjoint classification tasks with 20 classes per task.

For domain-IL experiments, we use PACS dataset (Li et al., 2017), which is widely used for domain generalization. It can present a more realistic domain-shift challenge than the toy-setting of PermuteMNIST (Kirkpatrick et al., 2017). Images in PACS come from seven classes and belong to four domains: Paintings, Photos, Cartoons, and Sketches. In Seq-PACS for CL, each task only focuses on one domain and the sequence of tasks is Sketches $\rightarrow$ Cartoons $\rightarrow$ Paintings $\rightarrow$ Photos (increasing the level of realism over time) (Volpi et al., 2021). Except for Seq-MNIST, where the number of training epochs for each task is 1, the number of training epochs for each task in all other datasets is set as 5, which is enough for CL.

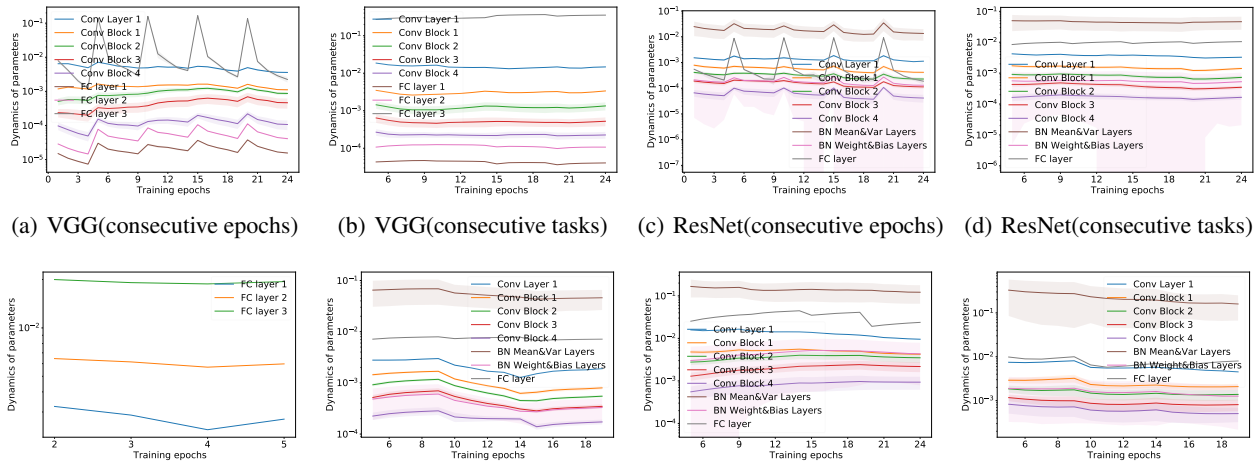
Models We follow the standard network architectures adopted in most previous CL works. For Seq-MNIST, following (Riemer et al., 2018), we employ an MLP, i.e., a fully-connected (FC) network with two hidden layers, each composed of 100 ReLU units. Following (Li et al., 2020; Derakhshani et al., 2022), we train ResNet-18 (He et al., 2016) on other five datasets. In addition, we also extend our empirical study to other architectures, like, VGG-11 (Simonyan & Zisserman, 2014) and ResNet-34 (He et al., 2016) on different datasets.

4. Forgetting of Different Parameters: An Empirical study

A fundamental and long-lasting question in CL is how the distribution shift impacts different model parameters and why it leads to harmful forgetting. Its answer could unveil the plasticity-stability trade-off in CL, where some parameters are plastic and task-specific and thus have to be fine-tuned before deploying the model, while the stable ones can be shared with and generalized to new tasks. Inspired by (Zhou et al., 2020; 2021), understanding the learning dynamics of deep neural networks is crucial for optimizing the training process. We conduct a comprehensive empirical study that compares the training dynamics of different parameters in three widely studied neural networks.

4.1. Measuring Forgetting via Training Dynamics

To measure and compare the forgetting effects of different parameters, we adopt two intuitive metrics to compute the change of parameters and investigate their dynamics over CL. The first metric calculates the difference between two consecutive epochs, e.g., for parameter θ_ℓ, it computes (1) $(1/|\theta_\ell|)\|\theta_{\ell,n}^t - \theta_{\ell,n-1}^t\|_1$ between epoch-$(n-1)$ and epoch-n within a task-t and (2) $(1/|\theta_\ell|)\|\theta_{\ell,1}^{t+1} - \theta_{\ell,N}^t\|_1$ between the last epoch of task-t and the first epoch of task-$(t+1)$. The training dynamics of this metric on different groups of parameters for different networks are shown in plots (a,c) of Fig. 1. (Refer to Sec. 4.2 for detailed analysis.) In CL, the unstable changes of parameters are mainly caused by the task shift, while the learning within each task usually leads to smooth changes. Hence, the second metric focuses on the difference between two consecutive tasks, e.g., the change of parameters between epoch-n of the two consecutive tasks, i.e., $C_\ell = (1/|\theta_\ell|)\|\theta_{\ell,n}^{t+1} - \theta_{\ell,n}^t\|_1$. Its results on different neural networks are displayed in plots (b,d,e) of Fig. 1.



(a) VGG(consecutive epochs) (b) VGG(consecutive tasks) (c) ResNet(consecutive epochs) (d) ResNet(consecutive tasks)

(e) MLP (f) Seq-OrganAMNIST (g) ER(buffer size= 2000) (h) Seq-PACS

Figure 1. (a-e) The training dynamics of different metrics for different groups of parameters when applying SGD in CL to train three types of deep neural networks on Seq-CIFAR-10; The training dynamics of other scenarios in ResNet-18: (e) on a non-standard dataset; (g) using a different CL method with a different buffer size; (h) in the domain-IL setting. Note the y-axis is of logarithmic scale.

(Refer to Sec. 4.2 for detailed analysis.)

4.2. Forgetting of Different Parameters During CL

We first investigate and compare the training dynamics of different parameters in three types of networks. To gain insights applicable to all CL methods, we exclude any specific CL techniques but simply apply SGD to train a model on a sequence of tasks without any countermeasure to forgetting. Then, we extend the experiment to different CL methods and datasets to verify whether the observations still hold. The number of tasks for each dataset and the number of training epochs for each task can be found in Sec. 3.

Dynamics between Consecutive Epochs Plots (a,c) of Fig. 1 show the training dynamics of consecutive epochs for parameters in VGG-11 and ResNet-18 when trained on Seq-CIFAR-10. We partition all parameters in VGG-11 into several groups, i.e., the bottom convolutional layer (closest to the input), convolutional layers in different blocks, and three FC layers. Besides the groups of VGG-11, ResNet-18 applies batch-normalization (BN), which has two groups of parameters, i.e., (1) weights and biases and (2) mean and variance. In the plots, all parameters experience more changes at the epoch of task switching and quickly converge after a few epochs in the same task. Hence, the dynamic patterns of this metric can be used to detect task boundaries.

Dynamics between Consecutive Tasks Plots (b,d,e) of Fig. 1 show the training dynamics of consecutive tasks for parameters in VGG-11, ResNet-18, and MLP. We train a three-layer MLP for Seq-MNIST. Since each task in Seq-MNIST is trained only 1 epoch, the dynamics of MLP for consecutive epochs and consecutive tasks are the same. From the plots of different neural networks, the last FC layer is more sensitive to task shift than other layers. When BN is included in the network, BN layers' mean and variance become the most changed parameters. These observations are similar to studies in research domains like multi-task learning and domain adaptation (Long et al., 2015; Chang et al., 2019) that the last FC layer and BN layers are task-specific and cannot be shared among tasks. In CL, the last FC layer is sensitive because tasks in class-IL differ on their predicted classes, which are the outputs of the last FC layer. It is also intuitive that BN layers are task-specific since the mean and variance of BN layers capture the first and second-order moments of the distribution for the latent representations. The variance of BN weight and bias is relatively large compared to other layers. Please refer to Appendix. G for details.

One interesting observation of VGG-11 and ResNet-18 is that the sensitivity of convolutional layers increases as the layer gets closer to the input. The reason may be that they are producing representations for the input images, whose distribution shift directly impacts the bottom convolutional layer. The functionality of filters in the top layers is to integrate patterns learned in the bottom layers to produce high-level patterns, so filters in the top layers are relatively stable. In Fig. 9 of the Appendix, we further study the training dynamics of each filter within a task or cross tasks in different layers of a network. Firstly, the training dynamics of each filter in the bottom layer are much larger than that of the top layer, which is in line with our above observation. We also find that in the same layer, when tasks shift, the dynamics of a small number of filters increase significantly. These filters should be task-specific and critical to the learning of the new task.

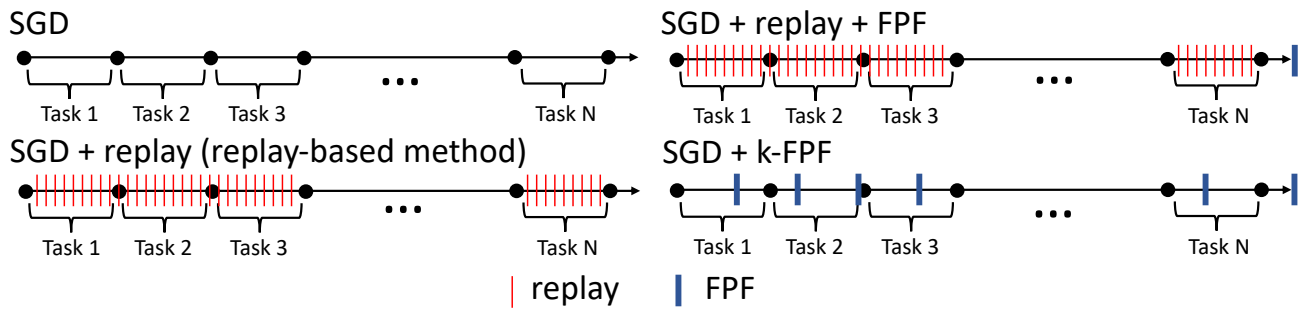


Figure 2. Comparison of SGD, replay-based method, FPF, and k-FPF. SGD trains tasks sequentially without replay. Replay-based methods train models on buffered and current data simultaneously. FPF finetunes the most sensitive parameters for a few iterations using buffered data at the end of arbitrary CL methods. k-FPF periodically (regardless of task boundaries) applies FPF for k times during training.

Dynamics on different scenarios The above empirical study is limited to SGD without applying any other CL techniques and only focuses on class-IL. In the following, we extend the studies to different CL methods, non-standard datasets, and domain-IL while fixing the model to be ResNet-18. Fig. 1 (f) extends the empirical study to a medical dataset Seq-OrganAMNIST. Compared to Seq-CIFAR-10, it differs in the number of tasks, dataset size, image size, and data type. We further replace SGD with ER using a replay buffer, whose results are reported in Fig. 1 (g). The ranking order of parameter groups in terms of sensitivity stays consistent under the change of dataset and replay strategy.

In domain-IL, as shown in Fig. 1 (h), the training dynamics of different parameters are in line with our observations in class-IL: only a small portion of parameters are task-specific. However, one difference is worth noting. Since the output classes stay the same across tasks and only the input domain changes, the last FC layer which is the most sensitive in class-IL, becomes equally or less sensitive than the bottom convolutional layer. Hence, the plasticity and stability of parameters are impacted by how close they are to the changed data distributions.

Inspiration from Empirical Studies. Above studies shed light on the improvements of CL methods. **(1)** We compare the sensitivity of different parameters in three types of deep neural networks and observe that only a small portion of them are much more sensitive than others. This implies that only finetuning these task-specific (or plastic) parameters may suffice to retain the previous tasks. **(2)** The dynamics between consecutive epochs show that all layers experience more changes when tasks shift, which can be used to detect task boundaries during CL. Knowing task boundaries is a critical prerequisite for lots of CL methods. The proposed metric makes the CL problem much easier, makes these methods more general and can contribute to better CL methods. **(3)** According to Fig. 9, in convolutional layers, only a small part of task-specific filters leads to a great change of dynamics when tasks shift. The regularization or isolation of these filters can improve the performance of CL.

5. Forgetting Prioritized Finetuning Methods

The above empirical study of the training dynamics on parameters immediately motivates a simple but novel method for CL, i.e., "forgetting prioritized finetuning (FPF)", which can be applied to any existing replay-based CL method. In the more efficient k-FPF, we further remove the every-step replay and any other CL techniques but simply apply k-times of FPF in SGD training. In Fig. 2, we provide an illustration that compares SGD, replay-based methods, and our proposed methods. At last, we propose a metric to automatically identify sensitive parameters in each neural network. See Appendix. A for the detailed procedure of FPF and k-FPF.

FPF to improve CL performance. FPF applies lightweight finetuning to the most task-specific parameters using the buffered data after the training of arbitrary CL methods. Hence, it is complementary to any existing CL methods as a correction step to remove their biases in the task-specific parameters by finetuning them on unbiased buffered data. Thereby, it can improve the performance of any existing CL methods without causing notably extra computation.

k-FPF to improve CL efficiency and performance. FPF is a simple technique that brings non-trivial improvement, but it is applied after the training of an existing CL method. Unfortunately, many SOTA CL methods require time-consuming replay in every step, which at least doubles the total computation. Since only a few parameters are sensitive during the task shift, can we develop a replay-efficient CL that replaces every-step replay with occasional FPF? We

propose k-FPF that applies FPF k times during CL as shown in Fig. 2. Without the costly experience replay, k-FPF can still achieve comparable performance as FPF+SOTA CL methods but only requires nearly half of their computation. We can apply k-FPF with any replay-free method, e.g., SGD, which is usually used as a lower bound for CL methods. We still maintain a small buffer by reservoir sampling, but it is only for FPF, so SGD never accesses it. We lazily apply FPF on the buffer after every τ SGD step (in total k times over $k\tau$ SGD steps) without knowing the task boundaries.

k-FPF-CE+SGD We propose two variants of k-FPF, i.e., k-FPF-CE+SGD and k-FPF-KD+SGD. k-FPF-CE+SGD uses the cross-entropy loss to update the sensitive parameters during each FPF. In this paper, k-FPF-CE refers to k-FPF-CE+SGD if not specified. The objective of FPF in k-FPF-CE is: $\min_{\Theta^\star} L(\Theta^\star) \triangleq \mathbb{E}_{(x,y)\sim B}[l_{CE}(y, f_\Theta(x))]$ where $\Theta^\star$ denotes selected groups of task-specific parameters, B refers to the buffered data and l_{CE} is the cross-entropy loss.

k-FPF-KD+SGD to further improve performance Inspired by DER (Buzzega et al., 2020), we further propose k-FPF-KD that introduces knowledge distillation (KD) (Hinton et al., 2015) to the objective in k-FPF-CE. In this paper, k-FPF-KD refers to k-FPF-KD+SGD if not specified. Same as DER, the pre-softmax responses (i.e. logits) for buffered data at training time are stored in the buffer as well. During FPF, the current model is trained to match the buffered logits to retain the knowledge of previous models. The objective of FPF in k-FPF-KD is: $\min_{\Theta^\star} L(\Theta^\star) \triangleq \mathbb{E}_{(x,y)\sim B}[l_{CE}(y, f_\Theta(x))] + \lambda\mathbb{E}_{(x,z)\sim B}[l_{MSE}(z, h_\Theta(x))]$ where z is the logits of buffered sample x, l_{MSE} refers to the mean-squared loss, $h_\Theta(x)$ computes the pre-softmax logits and λ is a hyper-parameter balancing the two terms. Compared to the computation of k-FPF-CE, the additional computation of k-FPF-KD is negligible.

Selection of sensitive parameters for FPF and k-FPF A key challenge in FPF and k-FPF is to select the task-specific parameters for finetuning. Examples of the training dynamics for different layers of various networks are shown in plots (b,d,e) of Fig. 1, and their ranking does not change over epochs. So we propose to select sensitive parameters for different neural networks according to their training dynamics in the early epochs. Specifically, for each neural network, its layers are partitioned into G groups as shown in Fig. 1, we calculate the sensitive score S_g for each group of layers in the neural network by

$$S_g = \frac{(1/|g|)\sum_{\ell\in g} C_\ell}{\sum_{g=1}^{G}(1/|g|)\sum_{\ell\in g} C_\ell} * G \tag{1}$$

where C_ℓ is the training dynamics mentioned in Sec. 4.1. We calculate the ratio of sensitivity for group g over all the G groups in the network. Since each network consists of a different number of parameter groups, we multiply G to rescale the sensitivity score.

In the experiments later, under different scenarios and on various benchmarks, we evaluate the performance of FPF and k-FPF when selecting different subsets of task-specific parameters. In a nutshell, finetuning parameters of higher sensitivity achieve more improvement, which is in line with our findings in empirical studies. FPF outperforms all baselines when parameter groups whose $S_g > 1$ are regarded as sensitive parameters and account for only $1.13\%, 0.32\%$ and 0.15% of the number of all parameters in MLP, VGG-11, and ResNet-18. For k-FPF, finetuning more parameters, i.e., the earlier convolutional layers, achieves the best performance. This is the price of removing replay, which halves the computational cost. We set the threshold of sensitivity score to 0.3 so that k-FPF achieves SOTA performance and only $12.40\%, 1.69\%$, and 24.91% of parameters in MLP, VGG-11, and ResNet-18 are regarded as sensitive parameters.

6. Experiments

In this section, to compare FPF and k-FPF with SOTA CL methods, we conduct our experiments mainly on ResNet-18. We apply FPF and k-FPF to multiple benchmark datasets and compare them with SOTA CL baselines in terms of test accuracy and efficiency. Besides, we also compare the performance of finetuning different parameters in FPF and k-FPF and show that finetuning a small portion of task-specific parameters suffices to improve CL. FPF improves SOTA CL methods by a large margin under all these scenarios, while k-FPF achieves comparable performance with FPF but is more efficient. Please refer to the Appendix for more results and analysis.

Implementation Details. We follow the settings in (Buzzega et al., 2020) to train various SOTA CL methods on different datasets, except training each task for only 5 epochs, which is more practical than 50 or 100 epochs in (Buzzega et al., 2020) for the streaming setting of CL. Since the epochs are reduced, we re-tune the learning rate and hyper-parameters for different scenarios by performing a grid search on a validation set of 10% samples drawn from the original training set. For both FPF and k-FPF, we use the same optimizer, i.e., SGD with the cosine-annealing learning rate schedule, and finetune the selected parameters with a batchsize of 32 for all scenarios. The finetuning steps for FPF and k-FPF are 300 and 100, respectively. We perform a grid search on the validation set to tune the learning rate and other hyper-parameters. Please refer to Appendix. N for the hyper-parameters we explored.

Baseline methods. We apply FPF to several SOTA memory-based CL methods: ER, iCaRL, A-GEM (Chaudhry et al., 2018), FDR (Benjamin et al., 2018), DER, and DER++. Be-

Table 1. Test accuracy (%) of baselines, FPF, and k-FPF. "-" indicates that the algorithm is not applicable to the setting. k-FPF-KD applies an additional knowledge distillation loss to the objective of k-FPF-CE. **Bold** and **Bold gray** mark the best and second best accuracy.

BUFFER	METHODS	CLASS-IL				DOMAIN-IL
		SEQ-ORGANAMNIST	SEQ-PATHMNIST	SEQ-CIFAR-10	SEQ-TINY-IMAGENET	SEQ-PACS
	JOINT	91.92±0.46	82.47±2.99	81.05±1.67	41.57±0.55	70.85±8.90
	SGD	24.19±0.15	23.65±0.07	19.34±0.06	7.10±0.14	31.43±6.39
	oEWC (SCHWARZ ET AL., 2018)	22.71±0.67	22.36±1.18	18.48±0.71	6.58±0.12	35.96±4.59
200	GDUMB (PRABHU ET AL., 2020)	61.78±2.21	46.31±5.64	30.36±2.65	2.43±0.31	34.16±3.45
	k-FPF-CE	75.21±2.03	**72.88±3.22**	57.97±1.53	13.76±0.72	60.70 ±2.81
	k-FPF-KD	**80.32±1.16**	**74.68±4.72**	58.50±1.03	**14.74±0.94**	63.15±1.19
	ER (RIEMER ET AL., 2018)	71.69±1.71	51.66±5.86	45.71±1.44	8.15±0.25	51.53±5.10
	FPF+ER	76.92±2.26	67.34±2.68	57.68±0.76	13.08±0.65	**65.16±1.97**
	AGEM (CHAUDHRY ET AL., 2018)	24.16±0.17	27.93±4.24	19.29±0.04	7.22±0.15	40.54±3.43
	FPF+AGEM	72.22±2.45	66.88±3.05	55.33±2.19	12.27±0.49	57.33±0.76
	iCARL (REBUFFI ET AL., 2017)	79.61±0.56	54.35±0.94	59.60±1.06	12.13±0.20	-
	FPF+iCARL	80.28±0.58	71.20±2.19	**63.36±0.91**	**16.99±0.37**	-
	FDR (BENJAMIN ET AL., 2018)	68.29±3.27	44.27±3.20	41.77±4.24	8.81±0.19	45.91±3.54
	FPF+FDR	76.10±0.87	70.06±2.78	51.91±2.77	11.52±0.72	57.17±1.31
	DER (BUZZEGA ET AL., 2020)	73.28±1.33	54.45±5.92	47.04±3.03	9.89±0.58	46.93±4.94
	FPF+DER	79.63±1.21	67.29±3.75	56.67±2.19	12.65±0.60	61.49±1.37
	DER++ (BUZZEGA ET AL., 2020)	78.22±2.05	62.00±3.79	59.13±0.81	12.12±0.69	55.75±2.02
	FPF+DER++	**80.99±0.91**	68.78±2.99	**61.69±0.97**	13.72±0.40	**65.28±1.02**
500	GDUMB (PRABHU ET AL., 2020)	73.29±1.82	63.55±5.62	42.18±2.05	3.67±0.25	43.29±2.53
	k-FPF-CE	81.28±0.71	76.72±1.94	64.35±0.87	19.57±0.37	65.90±0.72
	k-FPF-KD	85.16±0.67	**79.20±3.89**	66.43±0.50	**20.56±0.32**	**66.42±2.21**
	ER (RIEMER ET AL., 2018)	80.45±0.99	57.54±3.05	57.64±4.27	10.09±0.34	52.72±4.01
	FPF+ER	84.07±1.26	69.83±2.87	65.47±2.64	18.61±0.70	64.27±1.91
	AGEM (CHAUDHRY ET AL., 2018)	24.00±0.18	27.33±3.93	19.47±0.03	7.14±0.10	35.29±4.94
	FPF+AGEM	78.98±1.80	73.32±3.73	57.84±1.98	16.16±0.30	62.40±1.89
	iCARL (REBUFFI ET AL., 2017)	82.95±0.47	57.67±1.13	62.26±1.09	14.81±0.37	-
	FPF+iCARL	83.88±0.69	73.56±3.00	**67.75±0.67**	16.69±0.29	-
	FDR (BENJAMIN ET AL., 2018)	76.62±1.81	40.08±4.13	43.52±1.74	11.33±0.33	48.50±4.67
	FPF+FDR	82.32±0.91	73.64±3.85	63.09±0.81	17.10±0.35	65.39±1.83
	DER (BUZZEGA ET AL., 2020)	82.52±0.52	66.71±3.40	55.98±3.35	11.54±0.70	47.63±3.85
	FPF+DER	**85.18±0.39**	74.13±3.12	67.52±0.83	17.34±0.53	65.69±1.66
	DER++ (BUZZEGA ET AL., 2020)	84.25±0.47	71.09±2.60	67.06±0.31	17.14±0.66	57.77±2.54
	FPF+DER++	**85.40±0.26**	**77.37±1.32**	**69.08±0.92**	**20.17±0.35**	**66.89±1.32**

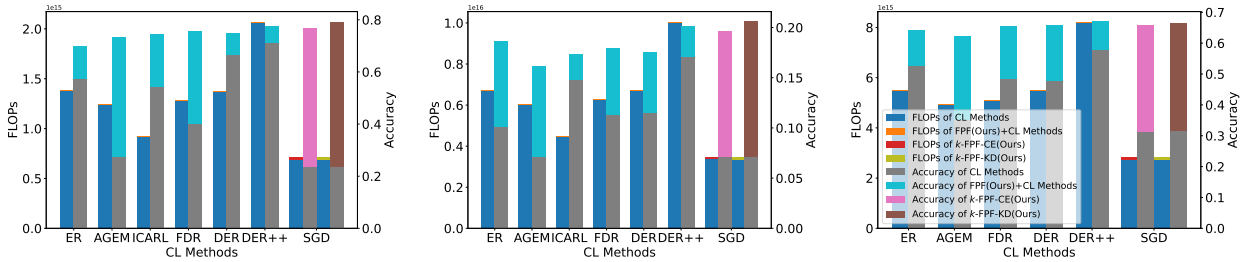
sides, we also compare our methods with GDUMB (Prabhu et al., 2020) and oEWC. We take JOINT as the upper bound for CL, which trains all tasks jointly, and SGD as the lower bound, which trains tasks sequentially without any countermeasure to forgetting. For FPF, k-FPF, and all memory-based methods, the performance with buffer sizes 200 and 500 is reported. All results reported in Tab. 1 are averaged over five trials with different random seeds.

6.1. Main Results

FPF considerably improves the performance of all memory-based CL methods and achieves SOTA performance over all scenarios in class-IL and domain-IL in Tab. 1. For methods with catastrophic forgetting, like AGEM, the accuracy of FPF increases exponentially. The surge in performance illustrates that FPF can eliminate bias by finetuning task-specific parameters to adapt to all seen tasks.

k-FPF-CE replaces the costly every-step replay with efficient occasional FPF. In Tab. 1, the performance of k-FPF-CE on Seq-PathMNIST, Seq-Tiny-ImageNet, and Seq-PACS is better than the best CL methods, and its performance on Seq-OrganAMNIST and Seq-CIFAR-10 is also better than most CL methods, which implies that finetuning the task-specific parameters on a small number of buffer during SGD can help retain the previous knowledge and mitigate forgetting, **every-step replay is not necessary**. In Fig. 3, the number of training FLOPs and the accuracy of different methods are reported. Compared to the training FLOPs of several CL methods, the computation cost of FPF and k-FPF-CE is almost negligible. The overall training FLOPs of k-FPF-CE are still much less than SOTA CL methods while its performance is better, which shows the efficiency of k-FPF-CE.

k-FPF-KD further improves the performance of k-FPF-CE to be comparable to FPF. k-FPF-CE proposes the efficiency of CL methods, but its performance is a bit worse than that of FPF. One of the most differences between k-



(a) Seq-PathMNIST (b) Seq-Tiny-ImageNet (c) Seq-PACS

Figure 3. Comparison of FLOPs and accuracy between FPF, k-FPF and SOTA methods. **FPF improves all CL methods by a large margin without notably extra computation. k-FPF consumes much less computation but achieves comparable performance as FPF.** A large and clear version can be found in Appendix. M.

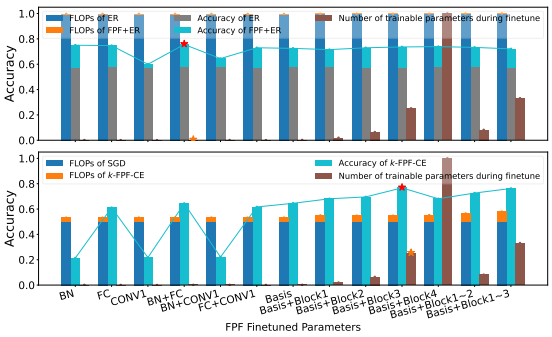


Figure 4. Comparison of FLOPs, number of finetuned parameters, and accuracy for FPF (Top) and k-FPF (Bottom) finetuning different combinations of parameters. All FLOPs are normalized together to (0,1], as well as the number of finetuning parameters. "Basis" in x-label refers to "BN+FC+CONV1". Red stars highlight the best accuracy and show **both FPF and k-FPF only require to finetune a small portion of task-specific parameters. k-FPF halves FPF's FLOPs.** A clear version can be found in Appendix. M.

FPF and FPF is the experience replay during the training of CL. Inspired by DER, we propose k-FPF-KD, which uses knowledge distillation to match the outputs of previous models on buffered data, hence retaining the knowledge of previous tasks. The results of k-FPF-KD in Tab. 1 show that it is comparable to FPF in most scenarios. Fig. 3 shows that the FLOPs of k-FPF-KD are similar to k-FPF-CE but much less than other CL methods and FPF, and in some cases, it outperforms FPF. **k-FPF-KD achieves SOTA performance on both efficiency and accuracy**.

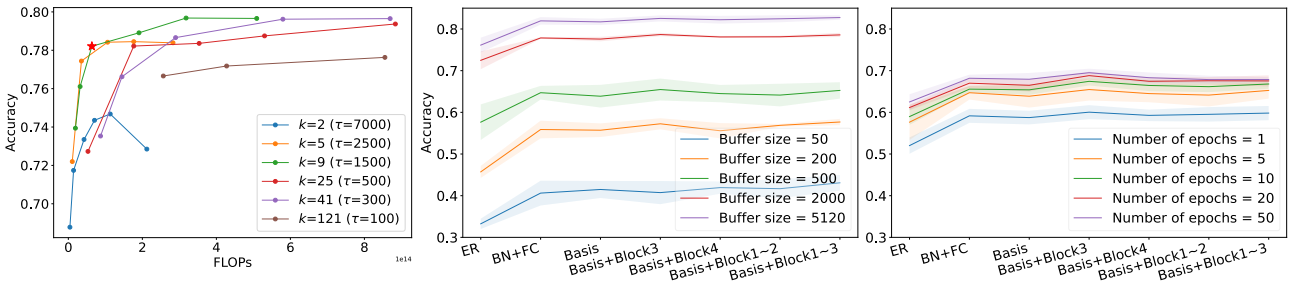
6.2. Comparison of finetuning different parameters in FPF and k-FPF

FPF and k-FPF get the best performance when only a small portion of task-specific parameters are finetuned. In Fig. 4, the accuracy, training FLOPs, and the number of trainable parameters during finetuning of applying FPF or k-FPF to different task-specific parameters in ResNet-18 on Seq-PathMNIST are compared. Overall different scenarios, k-FPF only needs about half FLOPs of FPF with better performance (indicated by Red Stars). When finetuning on different task-specific parameters, FPF performs the best when BN+FC layers are finetuned, which is only 0.127% of all parameters (indicated by Orange Stars). This is consistent with our observations in empirical studies where BN and FC layers are the most sensitive parameters to distribution shift. And the results show that only finetuning a small portion of task-specific parameters can mitigate catastrophic forgetting and generalize the model.

The phenomenon for k-FPF is a little different. (1) In the bottom plot of Fig. 4, when FC layer is not selected for finetuning in k-FPF, the performance is much worse. This is because, in class-IL, the output classes change across tasks, so the FC layer is trained to only output the classes for the current task (Hou et al., 2019). In contrast, when applying k-FPF to domain-IL on Seq-PACS, where the output classes keep the same for different tasks, Fig. 8 in Appendix. K shows that finetuning FC layer performs similarly as finetuning other parameters. Hence, the last FC layer is more sensitive in class-IL than in Domain-IL. This is also shown in Fig. 1 (d,h). (2) As the red star indicates, k-FPF needs to finetune a little more parameters (Block3 of convolutional layers, 18.91% of all parameters) to achieve a comparable accuracy with FPF. Without experience replay during SGD, the model has a larger bias on the current task, and thus more task-specific parameters are needed to be finetuned. This also indicates that such bias of task-specific parameters is the main reason for catastrophic forgetting. When Block4 (75.22% of all parameters) is finetuned, since it is the most stable group of parameters in our empirical study, the performance of k-FPF degrades.

6.3. Analysis of FPF and k-FPF in Different Scenarios

Different training FLOPs for k-FPF In Fig. 5 (a), we study the trade-off between the training FLOPs and the ac-



(a) FLOPs-Accuracy in k-FPF (b) Effect of different buffer sizes and training epochs during CL for FPF

Figure 5. **(a)** Trade-off between FLOPs and accuracy for k-FPF with different k and τ (the SGD steps between two consecutive FPF). By increasing the finetuning steps per FPF, the accuracy quickly saturates. The best trade-off is highlighted at the top-left corner when $k = 9(\tau = 1500)$. **(b)** Comparison between ER and FPF+ER finetuning different parameters with different buffer sizes and the number of epochs per task. In all scenarios, FPF can significantly improve the performance of ER by only finetuning BN+FC.

curacy of k-FPF on Seq-PathMNIST by changing k and the number of finetuning steps. τ in the legend refers to the interval of two consecutive FPF. Fixing k, k-FPF saturates quickly as the finetuning steps increase. This implies that k-FPF is efficient on FLOPs to achieve the best performance. For experiments with small k, e.g. k=2, though the computation required is very low, performance cannot be further improved. This implies that FPF needs to be applied on buffered samples more frequently to mitigate forgetting. When k is large, e.g., k=41 or 121, the accuracy slightly improves with the price of much more required computation. As the red star in the plot indicates, applying FPF for every 1500 training step can achieve the best computation-accuracy trade-off.

Different buffer sizes and training epochs for FPF The buffer size and the training epochs per task are usually crucial in replay-based CL methods. In Fig. 5 (b), when the buffer size or the number of epochs increases, the performance of ER improves as well. However, increasing the buffer size brings more benefits. When the buffer size or epochs grow too large, the performance of ER seems to saturate and increases slowly. For all scenarios, finetuning BN+FC layers is highly effective in alleviating the current task's bias and promoting performance, which is consistent with our observations from the empirical studies.

7. Conclusion

We study a fundamental problem in CL, i.e., which parts of a neural network are task-specific and more prone to catastrophic forgetting. Extensive empirical studies in diverse settings consistently show that only a small portion of parameters is task-specific and sensitive. This discovery leads to a simple yet effective "forgetting prioritized finetuning (FPF)" that only finetunes a subset of these parameters on the buffered data before model deployment. FPF is complementary to existing CL methods and can consistently improve their performance. We further replace the costly every-step replay with k-times of occasional FPF during CL to improve efficiency. Such k-FPF achieves comparable performance as FPF+SOTA CL while consuming nearly half of its computation. In future work, we will study how to further reduce the memory size required by FPF.

References

Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., and Tuytelaars, T. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 139–154, 2018.

Aljundi, R., Caccia, L., Belilovsky, E., Caccia, M., Lin, M., Charlin, L., and Tuytelaars, T. Online continual learning with maximally interfered retrieval. *arXiv preprint arXiv:1908.04742*, 2019.

Arani, E., Sarfraz, F., and Zonooz, B. Learning fast, learning slow: A general continual learning method based on complementary learning system. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=uxxFrDwrE7Y.

Benjamin, A. S., Rolnick, D., and Kording, K. Measuring and regularizing networks in function space. *arXiv preprint arXiv:1805.08289*, 2018.

Boschini, M., Bonicelli, L., Buzzega, P., Porrello, A., and Calderara, S. Class-incremental continual learning into the extended der-verse. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Buzzega, P., Boschini, M., Porrello, A., Abati, D., and Calderara, S. Dark experience for general continual

learning: a strong, simple baseline. *arXiv preprint arXiv:2004.07211*, 2020.

Caruana, R. Multitask learning. *Machine learning*, 28(1): 41–75, 1997.

Chang, W.-G., You, T., Seo, S., Kwak, S., and Han, B. Domain-specific batch normalization for unsupervised domain adaptation. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pp. 7354–7362, 2019.

Chaudhry, A., Ranzato, M., Rohrbach, M., and Elhoseiny, M. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

Derakhshani, M. M., Najdenkoska, I., van Sonsbeek, T., Zhen, X., Mahapatra, D., Worring, M., and Snoek, C. G. Lifelonger: A benchmark for continual disease classification. *arXiv preprint arXiv:2204.05737*, 2022.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hinton, G., Vinyals, O., Dean, J., et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

Hou, S., Pan, X., Loy, C. C., Wang, Z., and Lin, D. Learning a unified classifier incrementally via rebalancing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 831–839, 2019.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Kumari, L., Wang, S., Zhou, T., and Bilmes, J. A. Retrospective adversarial replay for continual learning. *Advances in Neural Information Processing Systems*, 35: 28530–28544, 2022.

Le, Y. and Yang, X. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Lesort, T., George, T., and Rish, I. Continual learning in deep networks: an analysis of the last layer. *arXiv preprint arXiv:2106.01834*, 2021.

Li, D., Yang, Y., Song, Y.-Z., and Hospedales, T. M. Deeper, broader and artier domain generalization. In *Proceedings of the IEEE international conference on computer vision*, pp. 5542–5550, 2017.

Li, D., Yang, Y., Song, Y.-Z., and Hospedales, T. Sequential learning for domain generalization. In *European Conference on Computer Vision*, pp. 603–619. Springer, 2020.

Long, M., Cao, Y., Wang, J., and Jordan, M. Learning transferable features with deep adaptation networks. In *International conference on machine learning*, pp. 97–105. PMLR, 2015.

Lopez-Paz, D. and Ranzato, M. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30:6467–6476, 2017.

Mallya, A. and Lazebnik, S. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 7765–7773, 2018.

Mermillod, M., Bugaiska, A., and Bonin, P. The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in psychology*, 4:504, 2013.

Pham, Q., Liu, C., and Hoi, S. Dualnet: Continual learning, fast and slow. *Advances in Neural Information Processing Systems*, 34:16131–16144, 2021.

Pham, Q., Liu, C., and HOI, S. Continual normalization: Rethinking batch normalization for online continual learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=vwLLQ-HwqhZ.

Prabhu, A., Torr, P. H., and Dokania, P. K. Gdumb: A simple approach that questions our progress in continual learning. In *European conference on computer vision*, pp. 524–540. Springer, 2020.

Ramasesh, V. V., Dyer, E., and Raghu, M. Anatomy of catastrophic forgetting: Hidden representations and task semantics. *arXiv preprint arXiv:2007.07400*, 2020.

Ratcliff, R. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285, 1990.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

Riemer, M., Cases, I., Ajemian, R., Liu, M., Rish, I., Tu, Y., and Tesauro, G. Learning to learn without forgetting by

maximizing transfer and minimizing interference. *arXiv preprint arXiv:1810.11910*, 2018.

Robins, A. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146, 1995.

Schwarz, J., Czarnecki, W., Luketina, J., Grabska-Barwinska, A., Teh, Y. W., Pascanu, R., and Hadsell, R. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, pp. 4528–4537. PMLR, 2018.

Serra, J., Suris, D., Miron, M., and Karatzoglou, A. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pp. 4548–4557. PMLR, 2018.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Van de Ven, G. M. and Tolias, A. S. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

Vitter, J. S. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1): 37–57, 1985.

Volpi, R., Larlus, D., and Rogez, G. Continual adaptation of visual representations via domain randomization and meta-learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4443–4453, 2021.

Wu, T., Caccia, M., Li, Z., Li, Y.-F., Qi, G., and Haffari, G. Pretrained language model in continual learning: A comparative study. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=figzpGMrdD.

Yang, J., Shi, R., Wei, D., Liu, Z., Zhao, L., Ke, B., Pfister, H., and Ni, B. Medmnist v2: A large-scale lightweight benchmark for 2d and 3d biomedical image classification. *arXiv preprint arXiv:2110.14795*, 2021.

Zenke, F., Poole, B., and Ganguli, S. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pp. 3987–3995. PMLR, 2017.

Zhang, C., Bengio, S., and Singer, Y. Are all layers created equal? *arXiv preprint arXiv:1902.01996*, 2019.

Zhou, T., Wang, S., and Bilmes, J. Curriculum learning by dynamic instance hardness. *Advances in Neural Information Processing Systems*, 33:8602–8613, 2020.

Zhou, T., Wang, S., and Bilmes, J. Curriculum learning by optimizing learning dynamics. In Banerjee, A. and Fukumizu, K. (eds.), *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pp. 433–441. PMLR, 13–15 Apr 2021. URL https://proceedings.mlr.press/v130/zhou21a.html.

A. Detailed procedure of FPF and k-FPF

The detailed algorithm of FPF, k-FPF and reservoir sampling are shown in Alg. 1, Alg. 2 and Alg. 3. For FPF, if the existing CL method belongs to replay-based methods, the buffer will be reserved according to this method. Otherwise, FPF will reserve the buffer by reservoir sampling. For both FPF and k-FPF, the data of each task may be trained for more than one epoch.

Algorithm 1 FPF

Input : Dataset D, an existing CL method M, early epochs iteration I, number of finetuning iterations K
Output : A well-trained CL model
Initialize : Buffer $B \leftarrow \emptyset$, training iterations $i = 0$

```
for (x, y) ∈ D do
    Run a batch of CL method M;
    i ← i + 1;
    if i = I then
        Identify sensitive groups of parameters {g} by Eq. 1.
    end
    B ← reservoir(B, (x, y)).
end
Finetune the sensitive groups of parameters {g} for K iterations on B.
```

Algorithm 2 k-FPF

Input : Dataset D, CL method SGD, early epochs iteration I, interval of FPF τ, number of finetuning iterations K
Output : A well-trained CL model
Initialize : Buffer $B \leftarrow \emptyset$, training iterations $i = 0$

```
for (x, y) ∈ D do
    Run a batch of SGD on (x, y);
    i ← i + 1;
    if i = I then
        Identify sensitive groups of parameters {g} by Eq. 1.
    end
    if i%τ = 0 then
        Finetune the sensitive groups of parameters {g} for K iterations on B.
    end
    B ← reservoir(B, (x, y)).
end
Finetune the sensitive groups of parameters {g} for K iterations on B.
```

Algorithm 3 RESERVOIR SAMPLING

Input : Buffer B, number of seen examples S, example x, label y
Output : An updated buffer B

```
if |B| > S then
    B[S] ← (x,y)
else
    k = randomInteger(min = 0, max = S));
    if k < |B| then
        B[k] ← (x,y)
    end
end
```

B. Forgetting and Task-IL accuracy

We show the forgetting and accuracy for training Seq-PathMNIST and Seq-PACS on ResNet-18 in Tab. 2 and Tab. 3. The results show that FPF also works well in Task-IL. The results of forgetting show that FPF is more stable than other CL methods and finetuning on buffered data can bring extra improvement.

Table 2. The results of forgetting and accuracy for training Seq-PathMNIST on ResNet-18

Methods	Class-IL Acc	Class-IL Forgetting	Task-IL Acc	Task-IL Forgetting
k-FPF-CE	76.72±1.94	**7.79±4.75**	97.34±0.13	0.23±0.62
k-FPF-KD	**79.20±3.89**	12.43±3.25	**97.57±0.28**	-0.49±0.57
ER	57.54±3.05	44.75±5.91	95.36±0.81	0.88±1.69
FPF+ER	69.83±2.87	16.84±2.36	96.84±0.16	-0.36±0.62
AGEM	27.33±3.93	92.87±4.22	96.33±1.47	-0.31±1.54
FPF+AGEM	73.32±3.73	20.79±2.69	96.85±0.99	0.60±1.26
iCaRL	57.67±1.13	41.90±3.19	75.15±2.11	28.62±2.73
FPF+iCaRL	73.56±3.00	10.24±4.69	97.2±0.29	**-0.58±0.71**
FDR	40.08±4.13	71.98±9.11	95.22±0.33	2.57±0.86
FPF+FDR	73.64±3.85	19.13±2.05	95.81±0.90	1.73±0.74
DER	66.71±3.40	36.67±4.64	95.61±0.69	1.96±0.81
FPF+DER	74.13±3.12	19.91±3.67	96.4±0.69	0.74±0.49
DER++	71.09±2.60	29.14±4.41	97.15±0.63	0.22±0.69
FPF+DER++	77.37±1.32	15.17±2.38	97.31±0.72	0.5±0.41

Table 3. The results of forgetting and accuracy for training Seq-PACS on ResNet-18

Methods	Domain-IL Acc	Domain-IL Forgetting
k-FPF-CE	65.90±0.72	**-13.83±2.74**
k-FPF-KD	66.42±2.21	-12.80±3.52
ER	52.72±4.01	3.23±3.27
FPF+ER	64.27±1.91	-7.69±5.90
AGEM	35.29±4.94	20.9±6.55
FPF+AGEM	62.40±1.89	-12.08±2.94
FDR	48.50±4.67	19.17±6.90
FPF+FDR	65.39±1.83	-1.42±2.68
DER	47.63±3.85	9.23±3.88
FPF+DER	65.69±1.66	-12.22±3.16
DER++	57.77±2.54	7.78±5.84
FPF+DER++	**66.89±1.32**	-0.87±2.83

C. Comparison between FPF and the method finetuning all parameters

In Tab. 4, we compare FPF with FPF-ALL (which finetunes all parameters) when applied to different CL methods for two types of CL, i.e., class-IL and domain-IL. The results show that FPF consistently achieves comparable or slightly higher accuracy than FPF-ALL by spending significantly fewer FLOPs. This demonstrates the advantage of FPF on efficiency.

Table 4. Comparison of accuracy and FLOPs between FPF and FPF-ALL(finetuning all parameters).

Methods	**Seq-PathMNIST**		**Seq-PACS**	
	Accuracy	FLOPs(B)	Accuracy	FLOPs(B)
k**-FPF-CE**	76.72±1.94	21.35	65.90±0.72	148.25
k**-FPF-ALL-CE**	75.74±2.91	43.95	64.48±2.23	174.60
FPF+ER	69.83±2.87	4.68	64.27±1.91	24.39
FPF-ALL+ER	70.64±4.00	8.79	63.81±2.33	34.92
FPF+AGEM	73.32±3.73	7.07	62.40±1.89	18.47
FPF-ALL+AGEM	74.80±3.12	8.79	62.65±1.65	34.92
FPF+iCaRL	73.56±3.00	4.27	-	-
FPF-ALL+iCaRL	72.77±4.12	8.79	-	-
FPF+FDR	73.64±3.85	2.94	65.39±1.83	11.70
FPF-ALL+FDR	74.24±1.48	8.79	64.88±2.28	34.92
FPF+DER	74.13±3.12	2.96	65.69±1.66	18.47
FPF-ALL+DER	74.54±3.19	8.79	66.22±0.87	34.92
FPF+DER++	77.37±1.32	4.68	66.89±1.32	24.39
FPF-ALL+DER++	77.16±1.45	8.79	65.19±1.33	34.92

D. Performance of various methods during the training of CL

In Tab. 5 and Tab. 6, the average test accuracy of previous tasks at the end of each task during the training of CL on Seq-PathMNIST and Seq-PACS is reported. The results show that during training, k-FPF can always achieve the best performance among various CL methods. Whenever the training stops, k-FPF can always achieve a model performing well on previous tasks.

Table 5. The average accuracy of previous tasks at the end of each task during the training of CL on Seq-PathMNIST.

Methods	Task 1	Task 2	Task 3	Task 4
k-FPF-CE	99.95±0.04	95.41±1.98	81.92±2.26	76.72±1.94
ER	98.62±1.59	83.06±3.12	74.60±3.18	57.54±3.05
AGEM	99.71±0.19	46.58±3.13	36.12±3.17	27.33±3.93
iCaRL	99.98±0.02	86.86±5.47	66.62±5.64	57.67±1.13
FDR	99.97±0.06	48.06±0.82	55.75±6.55	40.08±4.13
DER	99.98±0.02	91.92±3.42	76.50±5.77	66.71±3.40
DER++	99.95±0.06	94.06±6.14	80.35±3.32	71.09±2.60

Table 6. The average accuracy of previous tasks at the end of each task during the training of CL on Seq-PACS.

Methods	Task 1	Task 2	Task 3	Task 4
k-FPF-CE	70.94±2.02	73.75±2.68	62.37±0.49	65.90±0.72
ER	56.64±9.04	54.34±9.44	46.79±8.48	52.72±4.01
AGEM	47.34±7.35	38.02±5.82	32.70±7.13	35.29±4.94
FDR	58.59±4.36	54.00±4.01	46.38±4.80	48.50±4.67
DER	48.49±9.40	45.28±8.88	34.48±7.81	47.63±3.85
DER++	55.33±7.45	64.43±6.50	50.19±7.30	57.77±2.54

E. Evaluation of FPF and k-FPF on the larger dataset and backbone

We extend our experiments to include training Seq-Tiny-ImageNet on ResNet-34 with a buffer size of 500. We analyze the training dynamics of each module in ResNet-34 as for Seq-Tiny-ImageNet in Fig. 6. Our results indicate that similar to ResNet-18, the batch norm and FC layers in ResNet-34 are much more sensitive than other layers. We also observe a gradual decrease in dynamics from the bottom convolutional layers to the top layers.

In Tab. 7, we show the results of training Seq-Tiny-ImageNet on ResNet-34. We can find that k-FPF still outperforms other methods and FPF can also boost the performance of existing CL methods by fine-tuning a few sensitive parameters. The results show that our proposed method works well when applied to larger datasets and backbones.

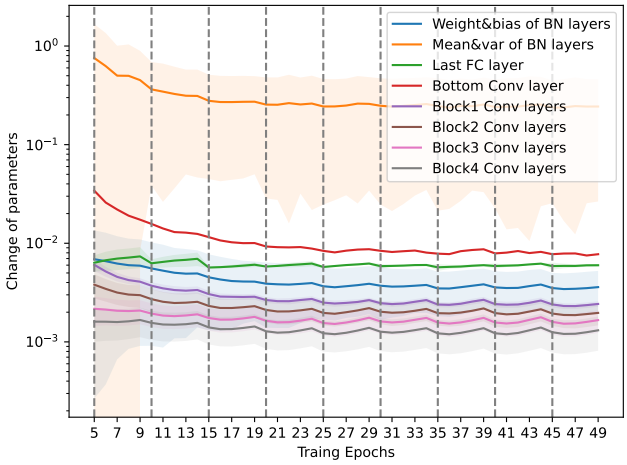


Figure 6. The training dynamics for different groups of parameters when applying ER in CL to train Seq-Tiny-ImageNet in ResNet-34. Note the y-axis is of logarithmic scale.

Table 7. The results of training Seq-Tiny-ImageNet on ResNet-34.

Methods	Class-IL Acc
k-FPF-CE	19.71±0.65
k-FPF-KD	**21.08±0.73**
ER	10.85±1.12
FPF+ER	15.70±0.72
AGEM	6.67±0.24
FPF+AGEM	16.69±0.47
FDR	13.88±1.03
FPF+FDR	16.50±0.25
DER	10.46±0.38
FPF+DER	16.35±0.36
DER++	16.33±1.26
FPF+DER++	18.87±0.39

F. Compare FPF with methods introducing additional backbones

We conduct the experiments of DualNet (Pham et al., 2021) and CLS-ER (Arani et al., 2022) by training Seq-CIFAR-10 on ResNet-18 with a buffer size of 500, the results are shown in Tab. 8. During the training of CL, both DualNet and CLS-ER introduce additional backbones to retain previous knowledge, so they perform better than other replay-based methods. Our method k-FPF-KD performs a little better than DualNet and CLS-ER. As a complement, our FPF can still greatly improve the performance of these two methods.

Table 8. The results of training Seq-CIFAR-10 on ResNet-18 using DualNet and CLS-ER.

Methods	Class-IL Acc
k-FPF-CE	64.35±0.87
k-FPF-KD	66.43±0.50
CLS-ER	66.10±0.73
FPF+CLS-ER	**70.42±1.22**
DualNet	65.47±0.89
FPF+DualNet	68.92±0.35

G. Detailed dynamics of BN weights and bias in different groups

In Fig. 7, the training dynamics of BN weights and biases in different groups are reported. This provides a fine-grained explanation of the phenomenon in Fig. 1 (d): the bottom BN layer is much more sensitive and task-specific than other BN layers. Consistent with convolutional layers, the deep BN layers are less sensitive to task drift than the shallower ones.

In a neural network, lower layers are closer to the input. Since the distribution of the inputs changes, the parameters of lower convolutional layers change sensitively to adapt to the distribution shift. The weights and biases of BN, which are the scale and shift of the featuremap, will change along with the convolutional parameters to adjust the distribution of the output featuremap. In the deeper layers, the functionality of each filter is relatively stable, so the distribution of the featuremap need not change drastically.

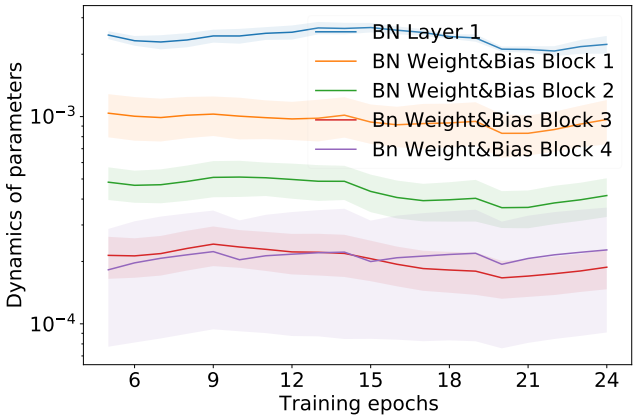


Figure 7. The training dynamics of different groups of BN weights and biases in ResNet-18.

H. Results of other neural networks

In Tab. 9, the results of various CL benchmarks and FPF on MLP and VGG-11 are reported. Similar to the results in Tab.1, by finetuning the most sensitive parameters in MLP and VGG-11, FPF can further improve the performance of all SOTA CL

methods and achieve the best performance. k-FPF-CE also achieves comparable performance as FPF + SOTA methods. Our methods can be generalized to various neural networks.

Table 9. Classification results for CL benchmarks and FPF on MLP and VGG-11. **Bold** and underline indicate the best and second-best algorithms in each setting.

BUFFER	METHODS	CLASS-IL	
		SEQ-MNIST(MLP)	SEQ-CIFAR10(VGG-11)
	JOINT	95.58±0.33	69.50±0.73
	SGD	19.64±0.07	18.71±0.33
	oEWC	20.69±1.34	18.46±0.23
500	GDUMB	90.60±0.37	41.65±0.78
	k-FPF-CE	90.63±0.57	55.45±1.16
	ER	86.73±1.03	46.27±1.18
	FPF+ER	91.15±0.16	53.48±1.08
	AGEM	51.03±4.94	19.40±1.09
	FPF+AGEM	89.26±0.52	29.84±1.37
	iCaRL	58.12±1.94	45.63±1.94
	FTF+iCaRL	80.83±0.49	48.03±0.65
	FDR	83.79±4.15	45.56±2.23
	FPF+FDR	89.67±0.37	55.59±1.56
	DER	91.17±0.94	51.12±2.47
	FPF+DER	**91.25±0.89**	**57,46±1.15**
	DER++	91.18±0.74	47.60±3.23
	FTF+DER++	91.22±0.67	54.69±0.73

I. Comparison between k-FPF, DER and DER++ with a large number of epochs for each task.

We compare the accuracy and FLOPs of our methods with the original results in (Buzzega et al., 2020) when allowing a large number of epochs on the same data for each task. The results are shown in Tab. 10. R-MNIST is a domain-IL dataset applied in (Buzzega et al., 2020). In both class-IL and domain-IL, k-FPF-CE is comparable to DER++ and k-FPF-KD is better than DER++ on the accuracy but spends much less FLOPs. These results demonstrate that our methods can outperform SOTA methods in various scenarios.

Table 10. Comparison of accuracy and FLOPs of k-FPF with the original results in (Buzzega et al., 2020) which has a large number of training epochs for each task.

Methods	Seq-CIFAR-10 Accuracy	Seq-CIFAR-10 FLOPs (B)	R-MNIST Accuracy	R-MNIST FLOPs (B)
k-FPF-CE	71.93±0.58	9208.85	91.15±0.29	0.64
k-FPF-KD	**74.32±0.32**	9208.85	**93.61±0.45**	0.64
DER	70.51±1.67	16726.26	92.24±1.12	1.29
DER++	72.70±1.36	25089.39	92.77±1.05	1.93

J. Comparison with related works (Ramasesh et al., 2020)

Paper "Anatomy of catastrophic forgetting: Hidden representations and task sementics" shows that freezing bottom layers had little impact on the performance of the second task. (i) Their setting is different: our study and most CL methods focus on the performance of ALL tasks. And it is unfair in terms of parameter amount to compare the freezing effects of multiple layers/blocks (e.g., block 1-3) vs. one layer/block. (ii) Their result is partially consistent with ours since their unfrozen part covers the last layer and many BN parameters, which are the most sensitive/critical part to finetune in our paper. (iii) The rest difference is due to our finer-grained study on parameters and on > 2 tasks, but this paper only studies two tasks and focuses on the second. Tab. 11 shows the class-IL accuracy at the end of each task if freezing different single ResNet blocks (bottom to top: block-1 to block-4). At the end of task 2, our observation is the same as this paper and freezing bottom blocks showed little reduction of accuracy. However, at the end of tasks 3-5, their performance drops, and freezing block-1 drops the most.

We also evaluate the importance of finetuning the first convolutional layer. We compare finetuning Basis (BN+FC+CONV1)

with BN+FC (without CONV1) and the results are reported in Tab. 12. It shows that removing CONV1 significantly reduces the accuracy and thus indicates the importance of finetuning CONV1.

Table 11. Class-IL accuracy of ER at the end of each task on Seq-CIFAR-10

	Task-1	Task-2	Task-3	Task-4	Task-5
No Freeze	97.52 ± 0.23	80.53 ± 0.80	63.96 ± 0.51	58.05 ± 1.91	57.03 ± 2.29
Freeze conv-1	97.52 ± 0.23	79.62 ± 2.75	63.28 ± 2.13	56.11 ± 0.61	55.58 ± 1.31
Freeze block-1	97.52 ± 0.23	78.88 ± 3.01	60.07 ± 0.61	55.49 ± 0.22	52.75 ± 1.90
Freeze block-2	97.52 ± 0.23	78.93 ± 3.34	63.78 ± 2.32	56.23 ± 0.82	56.55 ± 3.17
Freeze block-3	97.52 ± 0.23	80.37 ± 2.35	64.31 ± 2.23	57.21 ± 0.40	56.52 ± 0.76
Freeze block-4	97.52 ± 0.23	80.68 ± 1.53	64.89 ± 1.00	53.78 ± 3.37	54.01 ± 2.07

Table 12. The results of removing CONV1 from sensitive parameters.

Parameters	FPF+ER	k-FPF
Basis+Block1	71.53±2.34	68.10±1.93
BN+FC+Block1	68.99±4.67	67.80±3.19
Basis+Block2	72.98±2.62	69.57±0.90
BN+FC+Block2	68.12±2.51	69.60±1.40
Basis+Block3	73.64±3.85	76.72±1.94
BN+FC+Block3	70.33±4.85	75.46±2.75
Basis+Block4	73.88±2.42	68.12±2.40
BN+FC+Block4	73.72±4.73	68.35±1.56
Basis+Block1-2	73.28±2.42	73.33±1.34
BN+FC+Block1-2	69.25±2.52	71.46±1.88
Basis+Block1-3	72.11±1.55	76.22±2.75
BN+FC+Block1-3	71.22±1.49	74.18±1.84

K. Performance of finetuning different parameters for FPF and k-FPF on domain-IL dataset

In Figure 8, the performance of finetuning different parameters for FPF and k-FPF on domain-IL dataset Seq-PACS are reported.

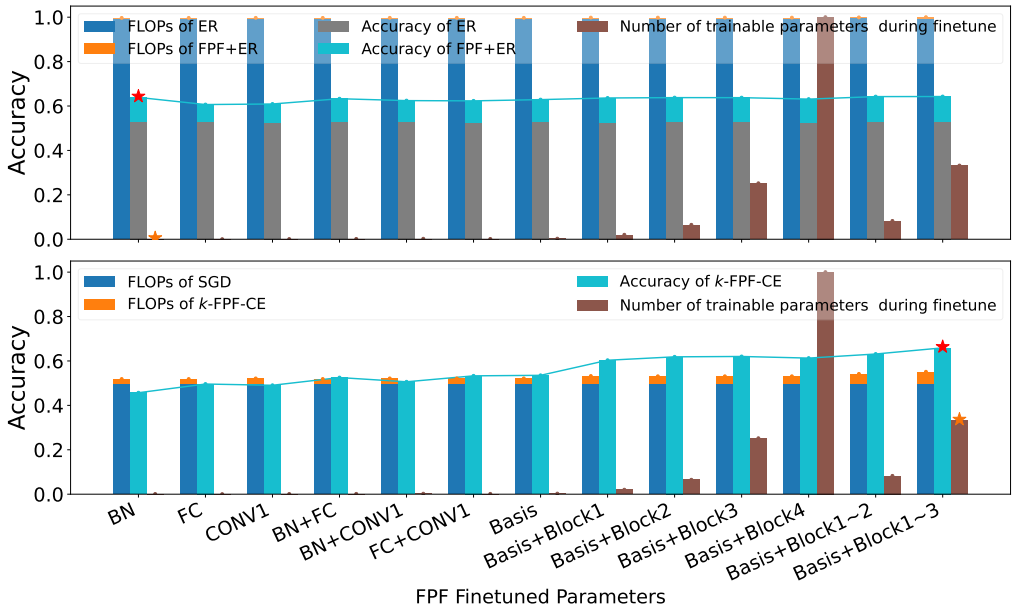


Figure 8. Comparison of FLOPs, number of finetuned parameters, and accuracy for FPF(Top) and k-FPF(Bottom) finetuning different combinations of parameters. All FLOPs are normalized together to (0,1], as well as the number of finetuning parameters. "Basis" in the x-label refers to "BN+FC+CONV1". Red stars highlight the best accuracy and show both FPF and k-FPF only require finetuning a small portion of task-specific parameters. k-FPF halves FPF's FLOPs. **Different from the results of k-FPF in class-IL, in Seq-PACS, since the output classes for different tasks are always the same, the last FC layer will not have a large bias on particular classes. Only finetuning BN or CONV1 layers for k-FPF can get comparable performance with ER.** Similar to class-IL, since experience replay is not allowed during the training of CL method SGD, a few more parameters are required to be finetuned by k-FPF to get comparable performance with FPF (about 24.92% of all parameters).

L. Distribution of filters' dynamics in different layers of neural networks

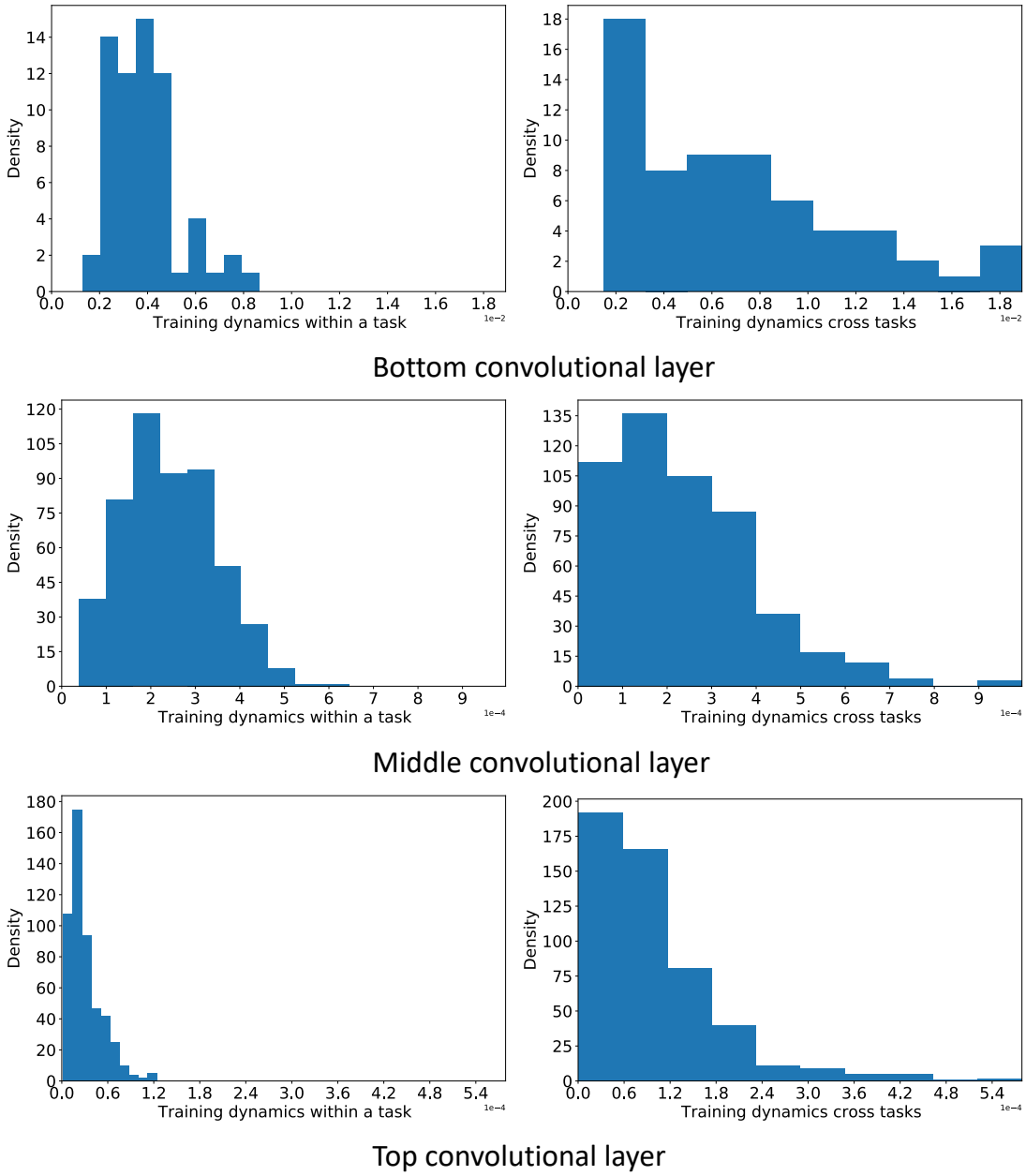
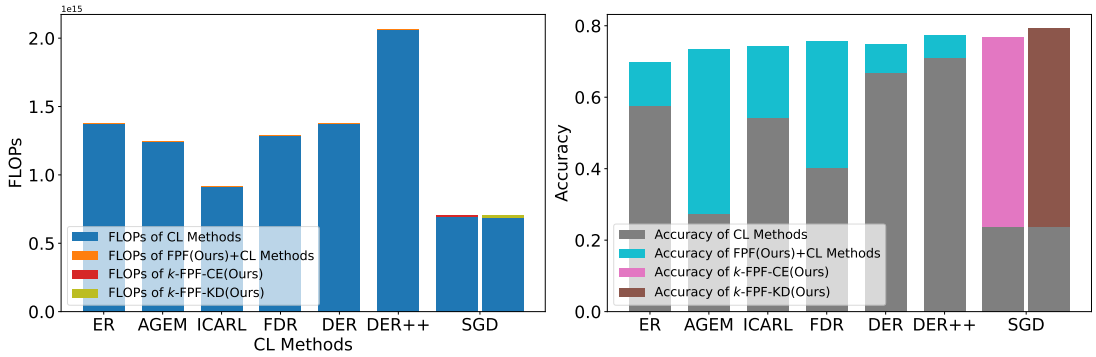


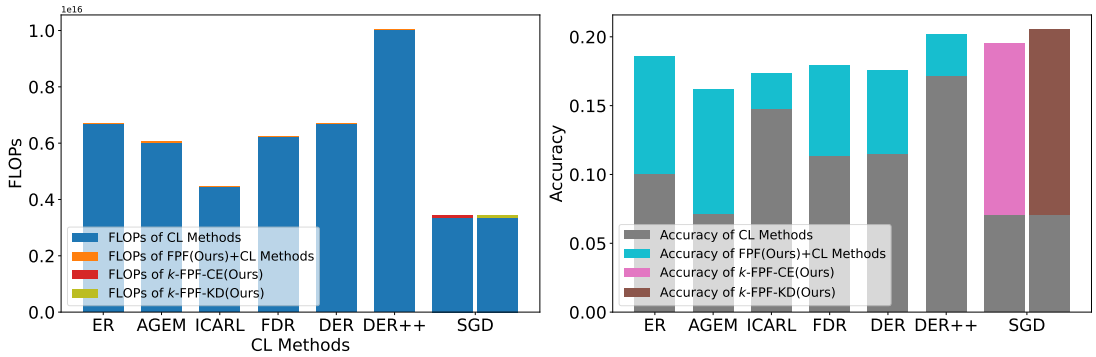
Figure 9. Comparison between the distribution of filters' training dynamics within a task and that cross tasks in different convolutional layers of VGG-11. When tasks shift, for all layers, only a small part of the filters in each layer experience more changes.

M. A more clear version of Fig. 3 and Fig.4

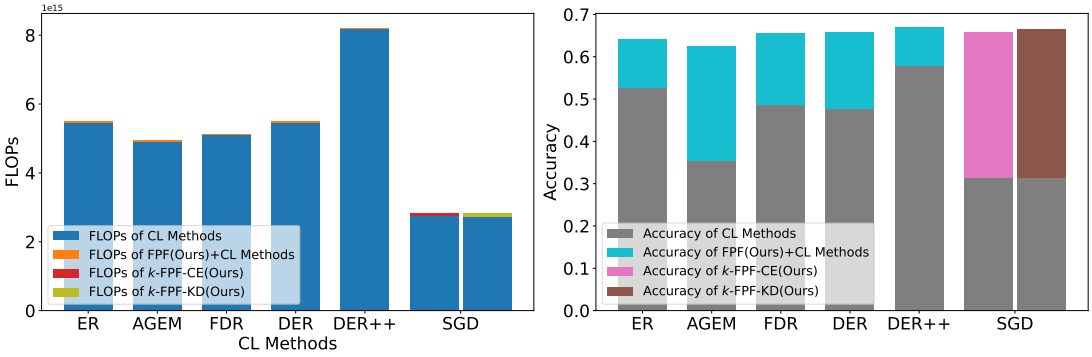
In Fig.10 and Fig.11, to make Fig.3 and Fig.4 more concise and easy to understand, we draw the bar plots of different parts separately.



(a) Seq-PathMNIST



(b) Seq-Tiny-ImageNet



(c) Seq-PACS

Figure 10. Comparison of FLOPs and accuracy between FPF, k-FPF and SOTA CL methods. **FPF improves all CL methods by a large margin without notably extra computation. k-FPF consumes much less computation but achieves comparable performance as FPF.**

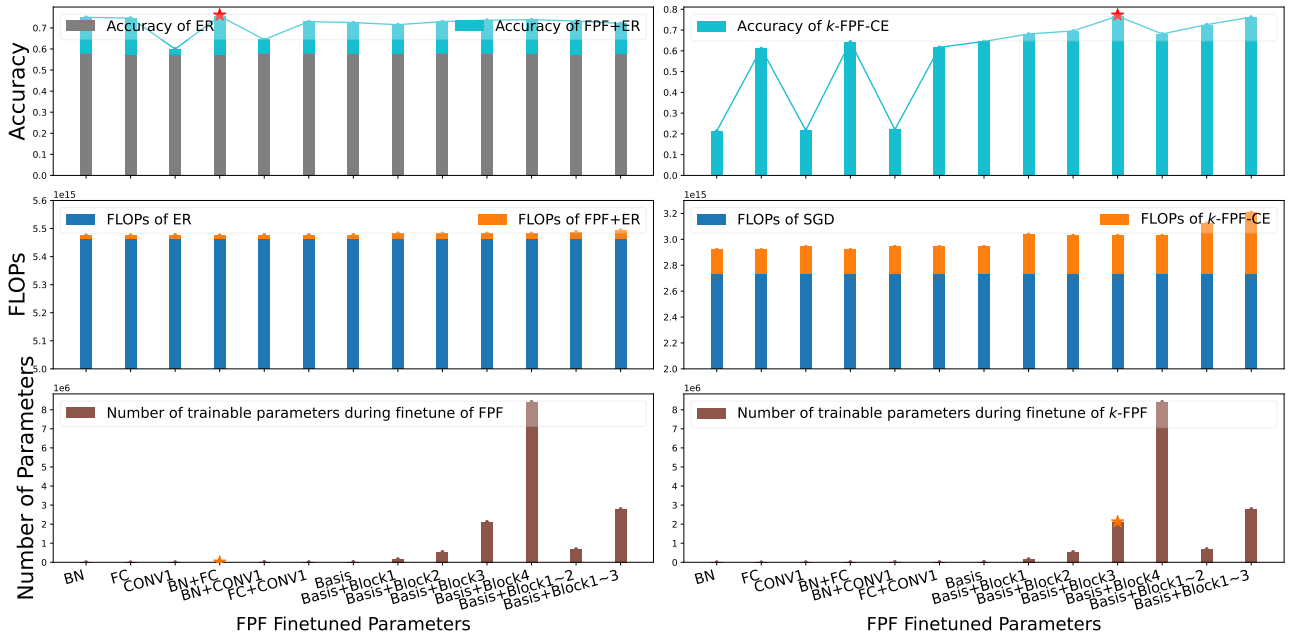


Figure 11. Comparison of FLOPs, number of finetuned parameters, and accuracy for FPF(Top) and k-FPF(Bottom) finetuning different combinations of parameters. All FLOPs are normalized together to (0,1], as well as the number of finetuning parameters. "Basis" in the x-label refers to "BN+FC+CONV1". Red stars highlight the best accuracy and show **both FPF and k-FPF only require to finetune a small portion of task-specific parameters. k-FPF halves FPF's FLOPs.**

N. Hyper-parameter Search Space

In the following, we provide a list of all the hyper-parameter combinations that were considered for FPF and k-FPF.

Table 13. The hyper-parameter search space for FPF on different datasets. For all experiments of FPF, we use the same number of batch size 32 and finetuning steps 300. The hyper-parameter spaces of finetuning different parameters in the models generated by different CL methods are always the same for a given dataset. ft-lr refers to the learning rate during finetuning of FPF.

Dataset	Hyper-parameter	Values
Seq-OrganAMNIST	lr	[1, 0.3, 0.1, 0.03, 0.01]
Seq-PathMNIST	lr	[1, 0.75, 0.3, 0.05, 0.03]
Seq-CIFAR-10	lr	[1, 0.3, 0.1, 0.03, 0.01]
Seq-Tiny-ImageNet	lr	[1, 0.5, 0.3, 0.075, 0.05]
Seq-PACS	lr	[1, 0.5, 0.3, 0.05, 0.03, 0.005, 0.003]

Table 14. The hyper-parameter search space for k-FPF-SGD on different datasets. For all experiments of k-FPF-SGD, we use the same number of batch size 32 and finetuning steps 100. The hyper-parameter spaces of finetuning different parameters are always the same for a given dataset. lr refers to the learning rate during training of CL method SGD. ft-lr refers to the learning rate during finetuning.

Dataset	Hyper-parameter	Values
Seq-OrganAMNIST	lr	[0.2, 0.15, 0.1, 0.075]
	ft-lr	[0.5, 0.2, 0.15, 0.1]
Seq-PathMNIST	lr	[0.05, 0.03, 0.01]
	lr	[0.1, 0.075, 0.05, 0.03, 0.01]
Seq-CIFAR-10	lr	[0.05, 0.03, 0.01]
	ft-lr	[0.075, 0.05, 0.03, 0.01]
Seq-Tiny-ImageNet	lr	[0.075, 0.05, 0.03]
	ft-lr	[0.1, 0.075, 0.05]
Seq-PACS	lr	[0.05, 0.03, 0.01]
	ft-lr	[0.075, 0.05, 0.03, 0.0075]

Table 15. The hyper-parameter search space for k-FPF-KD on different datasets. For all experiments of k-FPF-KD, we use the same number of batch size 32 and finetuning steps 100. The hyper-parameter spaces of finetuning different parameters are always the same for a given dataset. lr refers to the learning rate during training of CL method SGD. ft-lr refers to the learning rate during finetuning. λ is the hyper-parameter to balance the two losses.

Dataset	Hyper-parameter	Values
Seq-OrganAMNIST	lr	[0.2, 0.15, 0.1, 0.075]
	ft-lr	[0.5, 0.2, 0.15, 0.1]
	λ	[1, 0.5, 0.2, 0.1]
Seq-PathMNIST	lr	[0.05, 0.03, 0.01]
	lr	[0.1, 0.075, 0.05, 0.03, 0.01]
	λ	[1, 0.5, 0.2, 0.1]
Seq-CIFAR-10	lr	[0.05, 0.03, 0.01]
	ft-lr	[0.075, 0.05, 0.03, 0.01]
	λ	[0.5, 0.2, 0.1]
Seq-Tiny-ImageNet	lr	[0.075, 0.05, 0.03]]
	ft-lr	[0.1, 0.075, 0.05]
	λ	[1, 0.5, 0.2]
Seq-PACS	lr	[0.05, 0.03, 0.01]
	ft-lr	[0.075, 0.05, 0.03, 0.0075]
	λ	[1, 0.5 0.2 0.1]